Design of a View Support Subsystem of a Database System (Mathematical Methods in Software Science and Engineering : Second Conference)

AUTHOR(S):

MASUNAGA, YOSHIFUMI

CITATION:

MASUNAGA, YOSHIFUMI. Design of a View Support Subsystem of a Database System (Mathematical Methods in Software Science and Engineering : Second Conference). 数理解析研究所講究録 1980, 396: 64-103

ISSUE DATE:

1980-09

URL:

http://hdl.handle.net/2433/105026

数理解析研究所講究録
396巻 1980年 64-103

64

June 9-11,'80
at Kyoto Univ.

Design of a View Support Subsystem of a Database System

Yoshifumi Masunaga

Research Institute of Electrical Communication
Tohoku University
Katahira 2-1-1, Sendai, Miyagi 980
Japan

Abstract

In this paper, an architecture of implementing a view support subsystem of a relational DBMS is described by introducing the LUP(Local view Update Processor) concept with the view defining tree. That is, the LUP is a processor handling the translation of update, which initially stays at the root (i.e. the view) and then comes up step by step to a leaf (i.e. a base relation which is used to define the view) of the tree. If all LUP's in the tree succeed in reaching certain leaves, then the update execution will be initiated. The induced structural integrity constraint, the update modification rules and the augmentation rule are introduced as theoretical basis for describing the actions taken by a LUP. The actions of a LUP are described in detail, and the update execution control is also described. It is noted that the formal description of the meaning of a view is essential to define such LUP functions. Two examples are given to demonstlate this architecture. It should be stressed that the behavior of LUP's on the view defining tree just corresponds to a way of implementing the view update translation mechanism.

The Table of Contents

1. Introduction

Modern DBMS architecture tends to take the three level construction to provide the physical and the logical data independence as proposed in the ANSI/X3/SPARC report [TSIC78].

In this framework, there are external, conceptual and internal schema which describe the objects in the realms of interest according to the three levels. It is understood that the physical data independence will be achieved by specifying a mapping between the conceptual schema and the internal schema, while the logical data independence will be achieved by specifying a mapping between the external schema and the conceptual schema. However it is said that most of the present commercial or institutional DBMS's hardly provide any external schema views and therefore they provide almost no logical data independence [TSIC77, KIM79].

Usually the external schema consists of the virtual data which are defined in terms of the data in the conceptual schema. It is understood that one of the most serious reasons why the present DBMS's hardly provide any external schema views is that there are many difficulties in defining a mapping between the external schema and the conceptual schema.

In the relational data model, the conceptual schema consists of a set of relations, called base relations, and the external schema consists of relations, called (user's) views, which are derived from the base relations or other views by applying a sequence of the relational algebra operations and

the computing functions such as AVERAGE. Therefore the update to a view is only effected if the update to a view is translatable to the updates to the base relations which define the view and results the intended update result without causing any side-effect. There are at least two main issues in this problem. That is, the first one is to make it clear when and only when the update is translatable. The second one is to make it clear how the translation is done. Of course those two problems are closely connected.

In this paper, we try to make the second problem clear and give a solution to the first problem through the investigation to the second one. In the previous work (MASU79), we have investigated the first problem particuraly from the semantic point of view and showed that the meaning of a view should essentially be taken into account. Of course this paper stands on this point of view. However the main interest of this paper is to show how the second problem will be made clear from the algorithmic point of view. The new concept, LUP(Local view Update Processor) is introduced as a vehicle of implementing a view support subsystem, by which we can describe the mapping between the external schema and the conceptual schema. The LUP traverses the view defining tree from the lower level to the higher level and translates the update against the view step by step.

The following of this paper consists of as follows: In section 2, how views are defined is reviewed and the view

defining tree is introduced. In section 3, the traditional view update problems are shortly re-examined and see the point why the meaning of a view should be taken into account to discuss those problems. In section 4, the LUP is introduced and an architecture of implementing the view support subsystem is described using the LUP concept with the view defining tree. A few theoretical foundations are given, LUP functions are described in detail, and the update execution under the LUP concept is described. In section 5, two examples of update translation are demonstlated. Section 6 concludes this paper.

2. Views

2.1 Definition

In the relational data model, relationships among attributes of an entity set and the associations between entity sets are represented as relations. As defined in [CODD70], a relation R(A1, A2, ..., An) on n attribute domains dom(A1), dom(A2), ..., dom(An) (where dom(Ai) represents the domain of the attribute Ai,) is a finite subset of the direct product dom(A1) x dom(A2) x ... x dom(An). (We call this direct product the domain of R and denote it by dom(R).)

A relation which is physically realized on a certain storage device such as a disk is called a base relation. A set of base relations with integrity constraints consists of the conceptual schema of a relational database system. A view is a relation derived from the base relations (or other views) by applying a sequence of relational algebra operations and computing functions such as AVARAGE. (However, in this paper, we exclude views derived by computing functions to make our discussion simpler.) A view is a virtual relation and a set of views consists of an external schema of a relational database system.

2.2 View Defining Tree

In order to introduce the view defining tree, let us here review more precisely how views are defined: Originally the following eight operations are introduced as elements of the relational algebra, i.e. four traditional set operations

(the expanded direct product, union, intersection, difference) and other four less traditional operations on relations (projection, join, division, restriction) [CODD72]. However as it is known those eight operations are not mutually independent. Among them we deduce five operation, i.e. the expanded direct product ($\otimes$), union($\cup$), difference(-), projection(R[A]) and restriction(R[A θ B]), as a minimal set of operations. The reason of this selection is that the expanded direct product operation is essential to expand a relation, the projection and the restriction operations are essential to restrict a relation in the virtical and the horizontal direction respectively. It is clear that in this framework the θ-join of relation R on domain A with relation S on domain B is defined by $R[A \theta B]S = (R \otimes S)[A \theta B]$ and the division of R on A by S on B is defined by $R[A \div B]S = R[\bar{A}] - ((R[\bar{A}] \otimes S[B]) - R)[\bar{A}]$ [CODD72].

Now, the view defining tree is defined according to the defining expression of the view. For example, let us suppose that there are two base relations ED(EMP, DEPT) and DM(DEPT, MGR). In our framework, the natural join of ED on DEPT with DM on DEPT is defined by $ED \ast DM = ((ED \otimes DM)[DEPT^1 = DEPT^2])[EMP\ DEPT^2\ MGR]$, where the superscripted number 1 and 2 are used to distinguish that $DEPT^1$ belongs to ED and $DEPT^2$ belongs to DM.

The view defining tree of this view is shown in Fig.1. Notice that the notion of tree was found in [OSMA79].

3. View Update Problems

In this section we shortly review the traditional view update problems and see why the meaning of a view (or a relation) should be introduced to treate those problems. We do it by taking a simple but typical example:

Let the extensions of the base relation ED, DM and the natural join view EDM be as shown in Fig.2. (Those are denoted by ed, dm and edm respectively.)

(a) Suppose one wants to delete a tuple (e1, d1, m1) from the view edm. Then three alternatives are considerable for this update translation: (1) delete the pair (e1, d1) from ed, (2) delete the pair (d1, m1) from dm, (3) delete (e1, d1) and (d1, m1) from ed and dm respectively. But no alternative is adoptable without causing the side-effect.

(b) Suppose one wants to insert a tuple (e5, d3, m4) to edm. To effect this update, one might translate it into two insert statements, each of which inserts the pair (e5, d3) to ed and the pair (d3, m4) to dm respectively. But this translation causes the additive side-effect.

(c) Suppose one wants to delete a tuple (e3, d2, m3) from edm. In this case, any one of the following three alternatives is adoptable: (1) delete the pair (e3, d2) from ed, (2) delete the pair (d2, m3) from dm, (3) delete (e3, d2) and (d2, m3) from ed and dm respectively. But one can not decide uniquely which alternative should be chosen. This is one of the uniqueness problems.

Many investigations have been done to those problems [DATE71, CODD74, CHAM75, STON75, FERN76, PAOL77, DAYA78, BANC79, OSMA79, MASU79]. Among them, a semantic aspect of those problems was investigated in [PAOL77 , BAMC79 and MASU79]. Particularly, in [MASU79] the "meaning" of a relation was introduced and it played an essential role to characterize the translatable updates. (For example, the meaning of the base relation ED is a semantic nature of it by which we can see that "the employee e1 works in the department d1" as long as the pair (e1, d1) belongs to ED. This is formally denoted by M_{ED}.) The main results there were shown taking EDM as an example:

(a) Let M_{ED} and M_{DM} be the formal descriptions of the meaning of the base relations ED and DM respectively. Then the meaning of the natural join EDM is defined by

(E1) $(\forall t \in dom(ED) \times dom(DM))(M_{EDM}(t) \equiv M_{ED}(t[EMP \frown DEPT]) \wedge M_{DM}(t[DEPT \frown MGR]))$.

Now suppose one wants to delete a tuple (e1, d1, m1) from edm. Then the next statement holds:

(E2) "M_{EDM}(e1, d1, m1) is false if and only if either M_{ED}(e1, d1) is false or M_{DM}(d1, m1) is false or both."

If the intention of deleting the tuple (e1, d1, m1) from edm were comming from the fact that the pair (e1, d1) had lost the meaning (i.e. M_{ED}(e1, d1) is false), then the correct delete statement to be issued against edm should be a delete statement, which is capable of deleting all tuples having e1 and d1 as the EMP value and the DEPT value respectively. The

similar arguments hold for other two cases. Therefore the tuple delete requirement of deleting a tuple (e1, d1, m1) from edm was nonsense. As a result, this enables us to characterize the set of all translatable delete statements against EDM.

(b) So far as we are concern with the meaning of a view, the uniqueness problem stated previously does not arise.

(c) However, there exists yet another aspect of the view update problem. (This means that the previous additive side-effect comes from the (structural) nature of the natural join operation.) We note here that the meaning of a relation approach is still essential in the following investigations.

4. View Support Subsystem

4.1 LUP

LUP(Local view Update Processor) is a vehicle of implementing a view support subsystem of a relational DBMS. Generally, as observed in the previous section, the view update problems are complicated and therefore it seems very difficult to present a simple view update translation mechanism. However, it seems that one can possibly implement a view support subsystem if we first observe what happens when an update against a view is translated, and then identify some principles which rule the translation.

In our approach, a view is defined as a relation derived from the base relations by applying a sequence of five relational algebra operations and the derivation is shown as a tree which root represents the view, which leaves represent the base relations and which intermediate nodes represent certain intermediate relations. (We call the root is in the lowest level and therefore others are in higher level.) Therefore we can follow faithfully how the update against the view will be translatable to the updates to leaves by identifying when the translation is possible and how the translation is done, where we can find the concept of a processor which handles the translation. That is, the LUP is a processor which is allocated to a node, it governs the update translation at this node and can move on the tree. In our architecture, obviously an update is first issued against the root where a LUP stays initially. The general

form of (an) input to a LUP is a quadruple, the precise definition of which is given in section 4.3.1. The LUP output is the translated update statement(s) against its one level higher node(s) if the translation is possible and if not the announcement of the impossibility of translation. The LUP (if necessary the LUP is duplicated.) comes up to the higher node(s) when the translation was succeeded. The same action will be taken until all LUP's in the tree have reached to certain leaves. The precise description of the LUP actions is given in section 4.3.2 and 4.3.3. Next, if all LUP's in leaves succeed in executing updates, then they come down to the root where the expected update result will be obtained. This update execution is described in section 4.4 in detail. It should be noted here that the formal description of the meaning of a view is essentially taken into account in characterizing the LUP function.

As a summary, the LUP concept with the view defining tree is valid particuraly from the following points of view:

(a) The LUP concept with the view defining tree enables us to handle the view update translation just by looking at relations of one level higher and lower. This means that at most a finite number of actions of LUP's are definable (because only five relational operations are used to define the view defining tree and only two types of updates (deletion, insertion) are considered), while those actions are capable enough to handle the translation.

(b) LUP's process the view update translation step by step, which in turn means that the LUP concept with the view defining tree can handle the translation in a unified manner.

(c) The behavior of LUP's on the view defining tree just corresponds to a way of implementing the view update translation mechanism.

4.2 Theoretical Foundations

Before describing the function of LUP's in the following sections, we state a few theoretical foundations which are necessary to describe the function.

4.2.1 Induced Structural Integrity Constraint

Let us suppose that the expanded direct product of relation R(A1, A2, ..., An) and S(B1, B2, ..., Bp) is defined. Then the following integrity constraint should hold for $R \textcircled{x} S$:

(E3) $(\forall t, t' \in dom(R) \times dom(S))((t, t' \in R \otimes S) \supset ((t[A1 \frown A2 \frown \ldots \frown An] \frown t'[B1 \frown B2 \frown \ldots \frown Bp]) \in R \textcircled{x} S))$.

Obviously this is induced from the syntactic nature among tuples of the expanded direct product view. Therefore we call this the induced structural integrity constraint of the view.

Now, let us suppose that a tuple delete statement D is issued against the view. By res(D, $R \textcircled{x} S$), we denote the expected result relation. By diff(D, $R \textcircled{x} S$), we denote the set of all tuples of the view which should be deleted by D, i.e. $diff(D, R \textcircled{x} S) = R \textcircled{x} S - res(D, R \textcircled{x} S)$. To effect this tuple deletion, D should be translated into two tuple delete statements D_R and D_S (one of those may be empty) against R and

S respectively. (Here we do not want to say anything about how the translation will be done.) However, the point is that if D were effected, then the update result, res(D, R ⓧ S), should again satisfy the induced structural integrity constraint. That is, the following should hold:

(E4) $(\forall t, t' \in dom(R) \times dom(S))((t, t' \in res(D, R \otimes S)) \supset ((t[A1\frown A2\frown \ldots \frown An]\frown t'[B1\frown B2\frown \ldots \frown Bp]) \in res(D, R \otimes S)))$.

Now, the following is almost obvious:

Theorem

A delete statement D against R ⓧ S is translatable to the delete statement(s) against R and/or S (without causing any side-effect) if and only if (E4) holds.

We should note here that there is an exact correspondence between this theorem and the result of the characterization of the translatable delete statement D against R ⓧ S done from the semantic point of view [MASU79]. Because this investigation is valid in the following sections, a short summary is given below:

First, the meaning of R ⓧ S is formally defined by

(E5) $(\forall t \in dom(R) \times dom(S))(M_{R \otimes S}(t) \equiv M_R(t[A1\frown A2\frown \ldots \frown An]) \wedge M_S(t[B1\frown B2\frown \ldots \frown Bp]))$.

Second, the tuple delete statement D is issued against R ⓧ S, because every tuple of diff(D, R ⓧ S) has lost the meaning of R ⓧ S.

Then the following is obtained by (E5) and the particularization rule of the quantification theory:

(E6) "$M_{R \otimes S}(\text{diff}(D, R \otimes S))$ is false if and only if either (M_R(diff (D, R ⊗ S) [A1^A2^...^An]) is false or M_S(diff(D, R ⊗ S) [B1^B2^...^Bp]) is false or both."

Therefore the valid delete statement D against R ⊗ S should intend to delete either (a) any tuple of R ⊗ S which projection on A1^A2^...^An belongs to diff(D, R ⊗ S)[A1^A2^...^An], or (b) any tuple of R ⊗ S which projection on B1^B2^...^Bp belongs to diff(D, R ⊗ S)[B1^B2^...^Bp], or (c) both. It is now clear that a delete statement D against R ⊗ S satisfies the induced structural integrity constraint if and only if D is either one of the above three statements. Notice that if a delete statement satisfies this constraint, then no side-effect occurs.

The same argument holds for insert statements.

4.2.2 Update Modification Rule

Let V be a view which is associated with a certain intermediate node of a view defining tree. Moreover, let us suppose that (V is a direct product view and the one level lower relation of V is defined as a restriction V[A θ B] of it for certain A, B and θ .

Now, suppose a delete statement against V is D. Then one can modify D to D' such that (a) diff(D', V) ⊇ diff(D, V) and (b) diff(D', V) ∩ V[A θ B]) = diff(D, V). This is called a tuple delete statement modification rule. This modification is possible because V is an intermediate (therefore virtual) relation. However, in order not to cause any side-effect, the modification should be minimal (see section 4.3.2.1). An example of applying this rule is seen in example 5.1.

For a tuple insert statement I, one can modify I whenever V is a direct product view. The rule is called the insert statement modification rule. (The modification should also be minimal by the same reason.) A simple example of this case is given in Fig.3, where the tuple insert statement I of inserting the pair (3,3) to R ⊗ S is modified to the statement of inserting a set of pairs {(1,3), (2,3), (3,3)}, by which modification the insertion requirement of inserting a pair (3,3) to the view R[A=B]S is effected.

4.2.3 Augmentation Rule

This rule is used to modify an insert statement against a projection view. Suppose an insert statement I is issued against the projection view R[A]. As will be shown in section 4.3.3.4, in principle, I is not translatable to the insert statement against R because of the semantic ambiguity. However, when it is possible to determine u[$\bar{A}$](where u∈dom(R)) from res(I, R[A]) for any t in diff(I, R[A]) such that t = u[A], then we can translate I to an insert statement against R which realizes the insertion in R[A].

For example, the statement of inserting a tuple (e5, d3, m4) is translatable to the insertion against E[D=D]M because the $DEPT^1$ value is always determined by the $DEPT^2$ value. (That is, those two values are always equal.)

4.3 LUP Functions

In this section we describe how LUP's behave.

4.3.1 Preliminaries

We associate a distinguished non negative integer with

each node of a view defining tree (in a certain order). (An example is shown in Fig.1, where 0 is associated with the root and so on.) Generally, node n has one ancestor (i.e. the node of one level lower) except the root, and at most two descendant nodes (i.e. the nodes of one level higher). By anc(n), we denote the ancestor of node n, and by des(n), we denote the descendant of node n. (If there are two descendants, then by $des_L(n)$ and $des_R(n)$, we denote the upper left and the upper right node of node n.) By rel(n), we denote the relation defined at node n of the view defining tree. By reldef(n), we denote the defining expression of rel(n) in terms of its descendant relation(s). By U(n), we denote the update statement against rel(n), which is initially given by the user to the root and may be given by LUP's to the higher node(s).

General form of an input to a LUP is a quadluple (rel (anc(n)), rel(n), reldef(n), U(n)). For example, suppose a LUP stays at the root of the view EDM defining tree (Fig.1) and a delete statement D is issued against the view. Then the input to the LUP is (ϕ, edm, EDM=(E[D=D]M)[EMP$\frown$DEPT$^2\frown$MGR], D), where ϕ denotes an empty relation, i.e. the relevant node is the root.

The output of the LUP at node n is the translated update statement(s) against its one level higher node(s) if the translation is possible and if not the announcement of the impossibility of translation.

4.3.2 Deletions

Suppose a LUP stays at node n and U(n) is a delete statement D. The following states how the LUP functions in this case.

4.3.2.1 The Expanded Direct Product

Suppose $rel(n) = rel(des_L(n)) \otimes rel(des_R(n))$. Now, two types of inputs to LUP are considerable corresponding to (i) $anc(n) = \phi$ (i.e. n is the root) and (ii) $anc(n) \neq \phi$ (i.e. n is an intermediate node).

(i) Case of $anc(n) = \phi$

In this case, the input to the LUP is a quadruple $(\phi, rel(n), rel(des_L(n)) \otimes rel(des_R(n)), D)$.

Action D-1-1

"Check whether (E4) holds. If so, then output the translated update statements D_{Lout} and D_{Rout} to $des_L(n)$ and $des_R(n)$ respectively. (One of the outputs may be empty.) D_{Lout} and D_{Rout} are determined as stated in section 3. In this case the LUP moves to $des_L(n)$ ($des_R(n)$) whenever D_{Lout} (D_{Rout}) is not empty. (If both are non empty then the LUP is duplicated and those move to $des_L(n)$ and $des_R(n)$. Otherwise the LUP announces that the translation is impossible."

(ii) Case of $and(n) \neq \phi$

The input to the LUP is a quadruple $(rel(anc(n)), rel(n), rel(des_L(n)) \otimes rel(des_R(n)), D)$.

Action D-1-2

"Check whether (E4) holds. If so, then do the same as stated

in Action D-1-1. Otherwise, begin to apply the update modification rule (in section 4.2.2) to D so that the LUP may possibly find out D' which is a minimal modification of D. (Here the term "minimal" means that there does not exist any other modification D" of D such that res(D", rel(n)) satisfies (E4) and diff(D', rel(n)) $\supsetneq$ diff(D", rel(n)) $\supsetneq$ diff(D, rel(n)).) This condition is searched exhaustively. If such D' is found, then do the same as stated in Action D-1-1. Otherwise the LUP announces the impossibility of translation."

4.3.2.2 Union

Suppose rel (n) = rel(des_L(n)) $\cup$ rel(des_R(n)).

The formal description of the meaning of rel(n) is defined by

(E7) $(\forall t \in dom(rel(des_L(n))) \times dom(rel(des_R(n))))(M_{rel(n)}(t) \equiv M_{rel(des_L(n))}(t) \cup M_{rel(des_R(n))}(t))$.

Then the following holds:

(E8) "For any tuple t, $M_{rel(n)}(t)$ is false if and only if both $M_{rel(des_L(n))}(t)$ and $M_{rel(des_R(n))}(t)$ are false."

This means that to effect D against rel(n), all tuples of diff(D, rel(n)) should be deleted both from rel(des_L(n)) and rel(des_R(n)). Let D_{Lout} and D_{Rout} be the tuple delete statements against the left and the right descendant relation which deletes diff(D, rel(n)). Those two tuple delete statements are always definable.

Action D-2

"Output D_{Lout} and D_{Rout} to des_L(n) and des_R(n) respectively. The LUP is duplicated and those move to corresponding descend-

ant nodes."

4.3.2.3 Difference

Suppose $rel(n) = rel(des_L(n)) - rel(des_R(n))$.

The meaning of rel(n) is formally defined by

(E9) $(\forall t \in dom(rel(des_L(n))) \times dom(rel(des_R(n))))(M_{rel(n)}(t) \equiv M_{rel(des_L(n))}(t) \wedge \sim M_{rel(des_R(n))}(t))$.

Then,

(E10) "For any tuple t, $M_{rel(n)}(t)$ is false if and only if $M_{rel(des_L(n))}(t)$ is false or $M_{rel(des_R(n))}(t)$ is true or both."

This means that to delete a tuple t from rel(n), one can delete t from $rel(des_L(n))$ or insert t to $rel(des_R(n))$ or doing both simultaneously. There is no mathematical reason to decide which alternative should be chosen. However, notice that those have different meanings. That is, the first one means that t has lost the meaning of $rel(des_L(n))$, while the second one means that t becomes to satisfy the meaning of $rel(des_R(n))$. Therefore, essentially the LUP here can not choose arbitrarily and should ask to the user which one should be chosen.

Action D-3

"Ask to the user which alternative should be chosen. According to the answer, the LUP (if necessary it is duplicated) moves to the descendant node(s)."

However, if we do not want to have a LUP-user conversation, then we should give up to translate the delete statement:

Action D-3'

"Announce that the translation is impossible."

Notice again that the LUP should not be allowed to choose an alternative arbitrarily because it may cause semantic inconsistency.

4.3.2.4 Projection

Suppose rel(n)= rel(des(n))[A], where A is a list of attributes.

The meaning of rel(n) is formally defined by

(E11) $(\forall t \in dom(rel(n)))(M_{rel(n)}(t) \equiv (\exists u \in dom(rel(des(n))))$ $u[A]= t \wedge M_{rel(des(n))}(u)))$.

Then,

(E12) "For any tuple t, $M_{rel(n)}(t)$ is false if and only if for every u of dom(rel(n)), if u[A]= t then $M_{rel(des(n))}(u)$ is false."

This means that to delete a tuple t from rel(n), it is sufficiently enough to delete all tuple u of rel(des(n)) such that u[A]= t. In this case D is always translatable to D_{out} against rel(des(n)) straightforwardly so that it deletes all desired tuples. The LUP takes the following action:

Action D-4

"Output D_{out} as the delete statement against rel(des(n)) and the LUP moves to des(n)."

4.3.2.5 Restriction

Suppose rel(n)= rel(des(n))[A θ B], providing A and B are union-compatible.

The meaning of it is formally defined by

(E13) $(\forall t \in dom(rel(n)))(M_{rel(n)}(t) \equiv (t[A] \theta\ t[B]) \wedge M_{rel(des(n))}(t))$.

Then,

(E14) "For any tuple t, $M_{rel(n)}(t)$ is false if and only if, if $t[A] \theta\ t[B]$, then $M_{rel(des(n))}(t)$ is false.

This indicates the translation of D to the delete statement D_{out} against rel(des(n)), which is directly obtained by using the query modification method of [STON75]. The LUP action here is stated as follows:

Action D-5

"Output D_{out} as the delete statement against rel(des(n)) and the LUP moves to des(n)."

4.3.3 Insertions

Suppose a LUP stays at node n and U(n) is an insert statement I. The following states how the LUP functions in this case.

4.3.3.1 The Expanded Direct Product

Suppose $rel(n) = rel(des_L(n)) \otimes rel(des_R(n))$.

(i) Case of $anc(n) = \phi$

In this case, the input to the LUP is a quadruple (ϕ, rel(n), $rel(des_L(n)) \otimes rel(des_R(n))$, I). From the meaning point of view, the following is observed.

(E15) "For any t, $M_{rel(n)}(t)$ is true if and only if both $M_{rel(des_L(n))}(t[\alpha])$ and $M_{rel(des_R(n))}(t[\beta])$ are true."

This means that if we want to insert a tuple t to rel(n), then we should insert $t[\alpha]$ and $t[\beta]$ to $rel(des_L(n))$ and $rel(des_R$

(n)) respectively, where α and β denote the list of all attributes of rel(des_L(n)) and rel(des_R(n)) respectively.

Action I-1-1

"Check whether (E3) holds for res(I, rel(n)). If so, output I_{Lout} and I_{Rout} to des_L(n) and des_R(n) respectively, where I_{Lout} and I_{Rout} are the statements of inserting diff(I, rel(n))$\{\alpha\}$ and diff(I, rel(n))$\{\beta\}$ to rel(des_L(n)) and rel(des_R(n)) respectively. The LUP is duplicated and each moves to des_L(n) and des_R(n) respectively. If not, announce that the translation is impossible."

(ii) Case of anc(n)$\neq \phi$

In this case, the update modification rule is applicable.

The action of LUP is as follows:

Action I-1-2

"Check whether (E3) holds for res(I, rel(n)). If so, do the same as stated in Action I-1-1. Otherwise, begin to apply the update modification rule (in section 4.2.2) to I and find out I' which is the minimal modification of I. (The term minimal is defined analogously as did in Action D-1-2. In this case, such I' is always found.) Then do the same as stated in Action I-1-1."

4.3.3.2 Union

Suppose rel(n)= rel(des_L(n))$\cup$rel(des_R(n)).

In this case, the following holds from the meaning point of view:

(E16) "For any t, $M_{rel(n)}(t)$ is true if and only if either $M_{rel(des_L(n))}(t)$ or $M_{rel(des_R(n))}(t)$ or both are true."

This means that to effect I against rel(n), diff(I, rel(n)) should be inserted in either $rel(des_L(n))$ or $rel(des_R(n))$ or both. But notice that there is no mathematical reason which alternative should be chosen. This is completely a semantic issue as discussed already in the case of deletion against the difference view. (See section 4.3.2.3.)

Action I-2

"Ask to the user which alternative should be chosen. According to the answer, the LUP (if necessary duplicated) outputs the translated insert statement(s) and moves to the relevant descendant node(s)."

If we do not want to have such LUP-user conversation, the following is taken:

Action I-2'

"Announce that the translation is impossible."

Notice here that the LUP should not be allowed to choose an alternative arbitrary because it may cause semantic inconsistency.

4.3.3.3 Difference

Suppose $rel(n) = rel(des_L(n)) - rel(des_R(n))$.

By (E9) we obtain the following:

(E17) "For any t, $M_{rel(n)}(t)$ is true if and only if $M_{rel(des_L(n))}(t)$ is true and $M_{rel(des_R(n))}(t)$ is false."

This means that to effect I against rel(n), diff(I, rel(n)) should be inserted in rel(des_L(n)) and it should be deleted from rel(des_R(n)). The insert and the delete statement against those two relations respectively are definable straightforwardly. (We denote those by I_{Lout} and I_{Rout} respectively.)

Action I-3

"Output I_{Lout} and I_{Rout} to des_L(n) and des_R(n) respectively. The LUP is duplicated and each moves to the corresponding descendant node."

4.3.3.4 Projection

Suppose rel(n)= rel(des(n))[A], where A is a list of attributes. By (E11) we have the following:

(E18) "For any t, $M_{rel(n)}$(t) is ture if and only if there exists a tuple u of dom(rel(des(n))) such that u[A]= t and $M_{rel(des(n))}$(u) is true."

This means that to insert a tuple t in rel(n), the LUP should find out a tuple u satisfying (E18). However, there may not be possible to find out an unique u. The uniqueness is essential because different tuple represents different occurance of the entities and the relationships among them. Therefore, in principle, insert statements against the projection view is not translatable except the statement to which the augmentation rule is applicable. (See section 4.2.3.) In order to check whether the augmentation rule is applicable, the LUP should see the definition (i.e. intention) of rel(des(n)) in this case. Now the action of the LUP is made clear.

Action I-4

"Check whether the augumentation rule is applicable. If so, output the augmented insert statement I_{out} and moves to des(n). Otherwise, announce that the translation is impossible."

The problem here deeply relates to the null value issue in a relational data model (CODD75, ZANI77) which is another aspect of the view update problems (MASU79). But here we do not discuss this problem further.

4.3.3.5 Restriction

Suppose rel(n)= rel(des(n))(A θ B), providing A and B are union-compatible. By (E13), we obtain the following:

(E19) "For any t, $M_{rel(n)}(t)$ is true if and only if t(A) θ t(B) and $M_{rel(des(n))}(t)$ are true."

This indicates that we can translate I into the inserte statement I_{out}, which inserts diff(I, rel(n)) to rel(des(n)), according to the query modification method of (STON75).

Action I-5

"Output I_{out} to rel(des(n)) and moves to des(n)."

4.4 Update Execution

4.4.1 Execution Control

In section 4.1 and 4.3, we have described how LUP's behave. After a certain period of time, if all update translations are succeeded, then all LUP's stay at certain leaves. By the "LUP orbit", we mean a set of all paths, each of which begins at the root and ends at a certain leaf where a LUP reached. Then the LUP orbit consists a subtree of the view defining

tree. (For example, as shown in Fig.4(a), the LUP orbit is the straight line from node 0 to node 4, i.e. the set (0, 1), (1, 2), (2, 4) , in Example 1 of section 5.1.) If the LUP orbit is a straight line, then the update execution is straightforwardly done in such a way that first execute the update statement against the leaf relation, and then compute the extension of the view except using the new value of the leaf relation. However, if the LUP orbit is not a straight line but a proper subtree, then LUP's which stay at the upper nodes of a branching node should communicate each other to synchronize the execution of the update statement. For example, in Example 2 of section 5, LUP's at node 3 and 4 should be synchronized in the sense that the restriction operation EDDM $(DEPT^1 = DEPT^2)$ is executable after both LUP's come down to node 2. In order to realize the synchronization, we associate a "milestone" at each branching node of the LUP orbit. (In Example 2, as shown in Fig.4(b), node 2 has a milestone.) When a LUP first comes down to the node with a milestone, then the LUP should wait the pair LUP to come down the node. Except synchronization, the execution is done in ordinary manner.

4.4.2 Additive Side-effect Control

The additive side-effect may occur when one wants to insert a set of tuples to a certain view. In our framework, it may happen if there exists a LUP which took Action I-1-2 with the insert statement modification rule. However, we can observe that the rule is essentially necessary to effect a tuple

insertion to a certain view which is derived from an expanded direct product view. The modified insert statement inserts more tuples than those which are inserted by the original statement. The problem here is to investigate how the additionally inserted tuples interact with the view which is derived from the expanded direct product view. (In Example 2 of section 5.2, the quadruple (e4, d3, d3, m4) should be distinguished among the additionally inserted tuples, which causes the additive side-effect because it can pass the restriction EDDM(DEPT^1= DEPT^2).) As investigated in [MASU79], the additive side-effect issue is rather a structural issue than a semantic one in the sense that the additionally inserted tuple has correct meaning. Therefore, whether the use of the insert statement modification rule causes any additive side-effect should be checked exhaustively. That is, when a LUP uses the insert statement modification rule, the LUP associates a star mark(*) to the node (of the LUP orbit) to indicate the use of it. In update execution, LUP's should check whether the additive side-effect occur or not if they come down below the star marked node.

5. Examples

Let us now demonstrate how the update translations are done under the LUP concept with the view defining tree.

5.1 Example 1 -Deletion-

Suppose the view EDM is defined as shown in Fig.1 and the extensions are as given in Fig.2. Suppose a delet statement D_0 is issued against edm:

(E20) D_0: "Delete every tuple from edm having d1 and m1 as $DEPT^2$ and MGR value respectively."

Initially, a LUP stays at node 0 and then translate D_0 to D_1, which is the delete statement against e(d= d)m, and moves to node 1 (Action D-4):

(E21) D_1: "Delete every tuple from d(d= d)m having d1 and m1 as $DEPT^2$ and MGR value respectively."

Next, the LUP at node 2 decides to translate D_1 to D_2, which is the delete statement against eddm, and moves to node 3 (Action D-5):

(E22) D_2: "Delete every tuple from eddm having d1 and m1 as $DEPT^2$ and MGR value respectively and having the same $DEPT^1$ and $DEPT^2$ value."

The LUP at node 3, first examine whether (E3) holds for res(D_2, eddm). However the LUP seesthat it does not hold in this case. Then the LUP tries to apply the update modification rule (section 4.2.2) to D_2. In this case, the LUP succeeds in finding out such a statement which is D_3. (Notice that D_3 is obtained just by loosing the qualification part of D_2 such that the

condition of "having the same $DEPT^1$ and $DEPT^2$ value" is omitted)
(Update modification rule):

(E23) D_3: "Delete every tuple from eddm having d1 and m1 as $DEPT^2$ and MGR value respectively."

Then the LUP examines D_3 and translates it to the delete statement D_4 against dm and moves to node 4 (Action D-1-2):

(E24) D_4: "Delete every tuple from dm having d1 and m1 as $DEPT^2$ and MGR value respectively."

Now, the LUP recognized that it is in a leaf. Therefore it begins to execute D_4. As stated in section 4.4, the LUP comes down to the root where it can show the desired result.

5.2 Example 2 -Insertion-

Suppose the view EDM is defined as shown in Fig.1 and the extensions are as given in Fig.2. Suppose an insert statement I_0 is issued against edm(cf. section 3).

(E25) I_0: "Insert a tuple (e5, d3, m4) to edm."

The LUP stayed initially at node 0 translates it to I_1, which is an insert statement against e⌈d= d⌋m. This is possible because of the augumentation rule (section 4.2.3) and moves to node 1 (Action I-4):

(E26) I_1: "Insert a tuple (e5, d3, d3, m4) to e⌈d= d⌋m."

Then the LUP at node 1 translates I_1 to I_2, which is an insert statement against eddm, and moves to node 2 (Action I-5):

(E27) I_2: "Insert a tuple (e5, d3, d3, m4) to eddm."

Now the LUP first sees that (E3) does not hold for res(I_2, eddm). Therefor the insertion modification rule is used so that

I_2 is translated into two insert statements I_3 against ed and I_4 against dm:

(E28) I_3: "Insert a tuple (e5, d3) to ed."

(E29) I_4: "Insert a tuple (d3, m4) to dm."

As stated in section 4.2, node 2 is associated with a star mark. The LUP is duplicated and each of which comes up to node 3 and 4.

Now the LUP at node 3 and the LUP at node 4 begin to execute I_3 and I_4 respectively. Both LUP's come down and synchronized at node 2. Then two LUP's are marged into one, and execute the restriction EDDM$(DEPT^1 = DEPT^2)$. The LUP comes down to node 1 and because the upper node 2 is associated with the star mark, the LUP begins to check whether there exists an additionally inserted tuple which can pass the restriction. If there does not, then proceed execution, otherwise announce that the translation is impossible (The additive side-effect occurs.). In this case, as stated in section 4.4.2, the additionally inserted quadruple (e4, d3, d3, m4) passed the restriction. Therefore the translation of our insertion is impossible because of the additive side-effect.

6. Concluding Remarks

The LUP which is a vehicle of implementing a view support subsystem is introduced and the architecture of implementing the subsystem is described in detail. Through the investigation, the followings are observed with relation to the update translatability problem.

(a) In characterizing the actions taken by a LUP, the meaning of a view played an essential role. The ambiguity of the update translations is also characterized under it.

(b) The translatability and the translation mechanism of tuple delete statements are completely characterized from the meaning point of view. While this is not true for those of tuple insert statements. That is, the additive side-effect can not be controlled from this point of view, because this comes from the structural nature of the relational algebra. The null value issue closely relates to this problem.

(c) The LUP concept provides a very strong tool to distinguish such semantic and structural aspects of the view update problems.

Acknowledgement

The author is thankful to Prof. S. Noguchi who gave him an opportunity to work on this subject. This work is partly supported by the Science Foundation Grant of the Ministry of Education of Japan, Grant No. 468008.

Reference

[BANC79] Bancilhon, F., "Supporting View Updates in relational data base," Proc. IFIP TC-2 Working Conference on Data Base Architecture, 1979, pp. 198-219.

[CHAM75] Chamberlin, D.D., J.N.Gray and I.L. Traiger, "Views, authorization, and lockingin a relational database system," Proc. AFIPS NCC, 1975, pp. 425-430.

[CODD70] Codd, E.F., "A relational model of data for large shared data banks," CACM 13, 6, 1970, pp. 377-387.

[CODD72] Codd, E.F., "Relational completeness of database sublanguages," In Data Base Systems, Courant Computer Sci. Symp. 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, 1972, pp. 65-97.

[CODD74] Codd, E.F., "Recent investigations in a relational database system," Information Processing 74, North-Holland Pub. Co., Amsterdam, 1974, pp. 1017-1021.

[CODD75] Codd, E.F., "Understanding Relations," IBM Res. Report, July 10, 1975.

[DATE71] Date, C.J.and P.Hopewell, "File definition and logical data independence," Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, pp. 117-138 (1971).

[DAYA78] Dayal, U. and P.A.Bernstein, "On the updatability of relational views," Proc. Fourth VLDB Conf., 1978, pp. 368-377.

[FERN76] Fernandez, E.B. and R.C.Summers, "Integrity aspects of a shared data base," Proc. AFIPS NCC, 1976, pp. 819-827.

[KIM79] Kim, W., "Relational database systems," Computing Surveys 11, 3, pp. 185-211 (1979).

[MASU79] Masunaga, Y., "On a semantic aspect of view updates in a relational database system," prepared (1979).

[OSMA79] Osman, I.M., "Updating defined relations," Proc. AFIPS NCC, 1979, pp. 733-740.

[PAOL77] Paolini, P., and G.Pelagatti, "Formal definition of mappings in a database," ACM SIGMOD, Proc. of the Intl. Conf. on Management of Data, 1977, pp. 40-46.

[STON75] Stonebraker, M., "Implementation of integrity constraints and views by query modification," Proc. 1975 SIGMOD Conf., ACM, N.Y., pp. 65-78.

[TSIC77] Tsichritzis, D.C. and F.H.Lochovsky, "Data Base Management Systems (book)," Academic P., 1977.

[TSIC78] Tsichritzis, D.C. and A.Klug (Ed.)," The ANSI/X3/SPARC DBMS framework, report of the study group on database management systems," Inform. Systems 3, pp. 173-191 (1978).

[ZANI77] Zaniolo, C., "Relational views in a data base system support for queries, "IEEE COMPSAC 77, 1977, pp. 267-275.

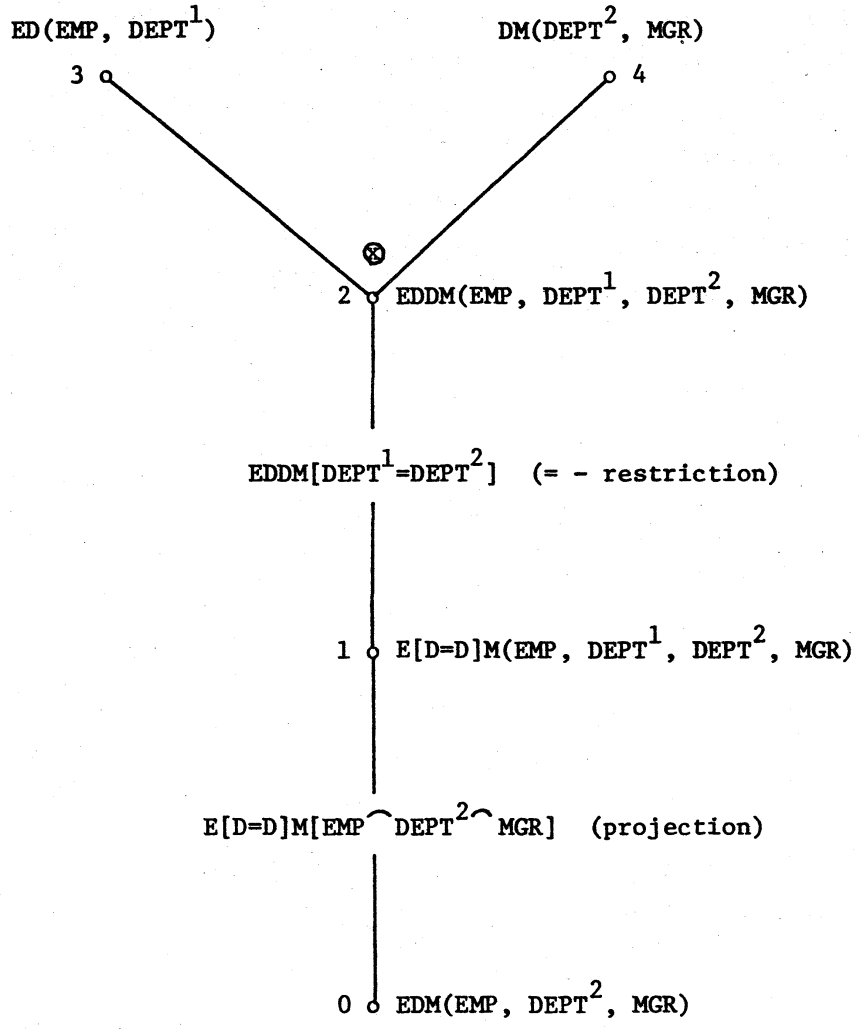


Fig. 1. The view defining tree of the view EDM.

ED

ed: EMP	$DEPT^1$
e1	d1
e2	d1
e3	d2
e4	d3

DM

dm: $DEPT^2$	MGR
d1	m1
d1	m2
d2	m3

EDDM

eddm: EMP	$DEPT^1$	$DEPT^2$	MGR	
e1	d1	d1	m1	*
e1	d1	d1	m2	*
e1	d1	d2	m3	
e2	d1	d1	m1	*
e2	d1	d1	m2	*
e2	d1	d2	m3	
e3	d2	d1	m1	
e3	d2	d1	m2	
e3	d2	d2	m3	*
e4	d3	d1	m1	
e4	d3	d1	m2	
e4	d3	d2	m3	

e[d=d]m: This consists of five tuples of eddm marked *.

EDM

edm: EMP	$DEPT^2$	MGR
e1	d1	m1
e1	d1	m2
e2	d1	m1
e2	d1	m2
e3	d2	m3

Fig. 2. Extensions of ED, DM, EDDM, E[D=D]M and EDM.

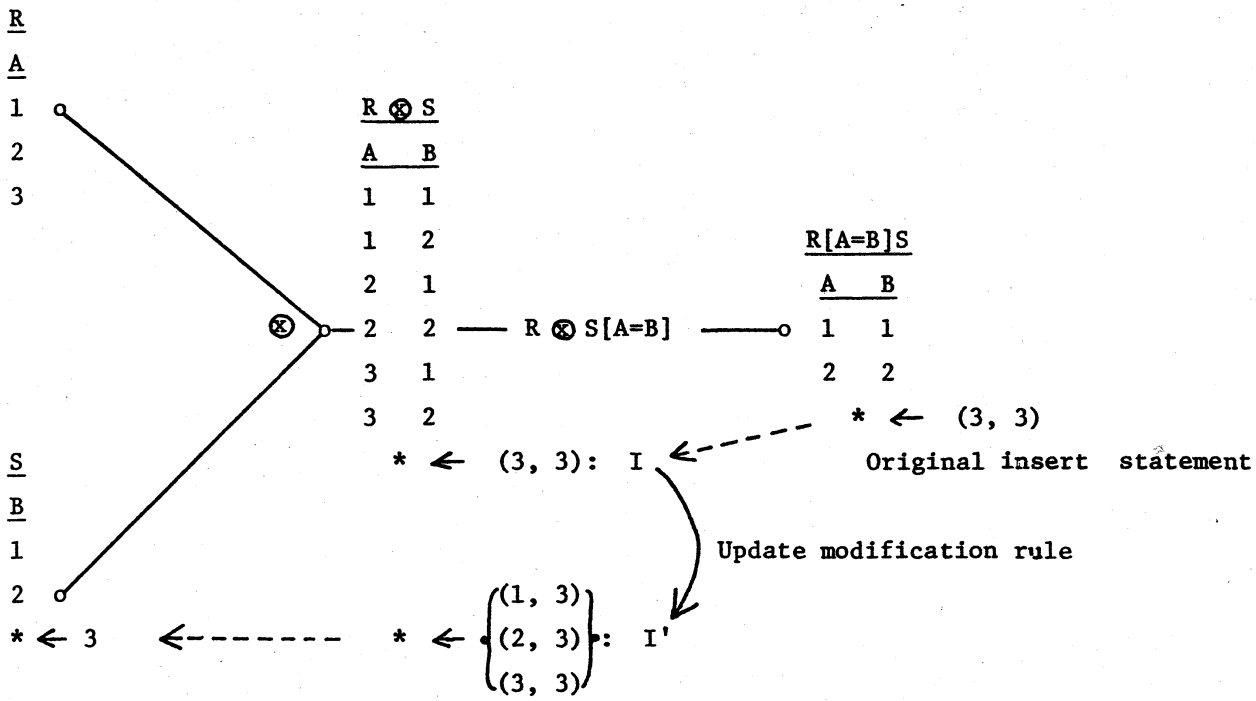
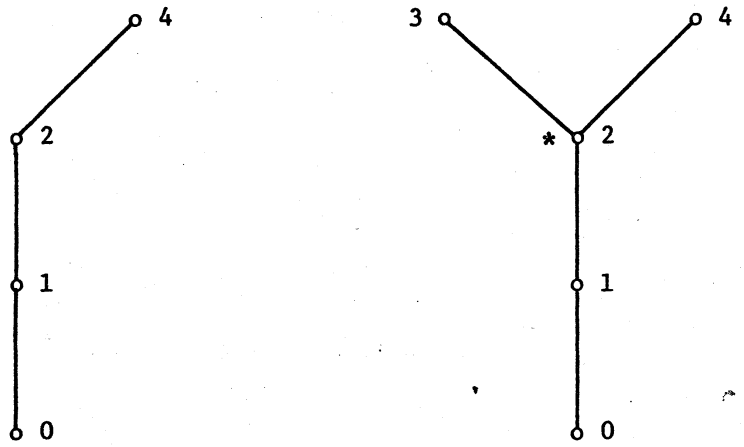


Fig. 3. Update modification rule.



(a) Case of Example 1. (b) Case of Example 2.

Fig. 4. The LUP orbits.